UCRL-WEB-218121

Environment Variables

Table of Contents

Preface

Scope:	This Environment Variables guide explains the role and use of environment variables on UNIX computing systems in general and on LC machines in particular. It surveys relevant tools, syntax, and security issues, then summarizes LC-relevant environment variables by what they do for users (system management, application support, batch-job enabling, etc.). LCRM's handling of environment variables, including those used for job control and checkpointing by SLURM, receives special attention, along with the many environment variables that SLURM uses to manage tasks and their resources. A later "dictionary" section cross references locally important environment variables in one alphabetical list.
Availability:	When the variables described here are limited by machine, those limits are included in their explanation. Otherwise, they appear on any LC UNIX system.
Consultant:	For help contact the LC customer service and support hotline at 925-422-4531 (open e-mail: lc-hotline@llnl.gov, secure e-mail: lc-hotline@pop.llnl.gov).
Printing:	The print file for this document can be found at:

```
OCF: http://www.llnl.gov/LCdocs/ev/ev.pdf
SCF: https://lc.llnl.gov/LCdocs/ev/ev_scf.pdf
```

Introduction

Environment variables are a standard aspect of any UNIX computing environment. The shells, your scripts and applications, and the job-management system (LCRM and SLURM) all rely on environment variables to both consistently share a software context across users and time and to customize that shared context (as allowed) to adapt it to your specific computing needs.

Many LC user manuals (and vendor documents too) discuss particular environment variables for particular purposes (MPI support, code threading, job control, output management). This guide aims to supplement those separate, narrow discussions with a general, comparative look at how environment variables contribute to your use of LC computing resources. It is not truly comprehensive, however, since hundreds (perhaps thousands) of environment variables are available on AIX and Linux systems, and we focus here on those most relevant to and most useful for successful high-performance computing on LC machines. The goal is not to replace other focused treatments of specific environment variables, but to support them with an organized, integrated overview that will help you more easily discover and more fully appreciate those technical details.

This manual begins with a functional approach. One section (page 5) summarizes the role of, standard syntax for, relevant software tools for, and some known security concerns with environment variables at LC. Another section (page 10) then revealingly groups and explains LC's most important environment variables by how they work on local machines (system support, user support, batch support, etc., with some overlap of course). Because the situation is complex, with many important consequences, LCRM's use of environment variables for job management receives especially thorough attention (page 27), as does the suite of environment variables that SLURM uses to control tasks and their resources on each cluster (page 37).

Later the text offers an alternative, alphabetical (or dictionary) approach (page 44). Here you can find an environment variable by name in one master list, and then link to a contextualized discussion of its functional background or intended role(s) elsewhere in the text.

Environment Variable Background and Philosophy

Roles

That variables serve as named local storage locations for string or numeric values is well known. Typical script examples include:

```
CSH--
     set myvar = this.string
     echo $myvar

SH--
     mvar=this.string
     echo "$myvar"
```

But the values of such ordinary variables are unknown to other processes or to the shell. They are strictly local to the process that declares them.

UNIX *environment* variables, however, are placed in the calling environment of every child process (each child process has a copy of these variables for its own use). When such environment variables are inherited by the shell or declared and initialized at shell startup, they apply to your login environment, to all other subshells later spawned, and to all the programs that run within those contexts. Hence, environment variables are a convenient way to share crucial constraints (your terminal type), preferences (your search path), or support information for reliable job execution (where needed libraries reside).

Each standard environment variable at LC:

- has a specified name (see the next subsection for syntax restrictions),
- stores a string value (which might include digits),
- may be preset for you (to standardize the computing environment and improve reliability),
- can be reset by you (sometimes needed or desirable, sometimes quite undesirable),
- helps create, remember, and share a common software context for your scripts and applications (which path to search, where to send output), and
- persists as long, but only as long, as you stay logged in (so customized values need to be redeclared in your "dot files" (.login, .profile, etc.) to execute at each login).

Syntax

Because environment variables are intended for widely sharing their contents among shells, UNIX utilities, and applications, some rules are necessary to make that sharing reliable. Thus IEEE Standard 1003.1-2001 specifies that:

- For shells and utilities, UNIX environment variable names should consist solely of
    - ◊ uppercase letters,
    - ◊ digits, and
    - ◊ the underscore (_) character.
- The namespace of environment-variable names containing *lowercase* letters is reserved for applications. To avoid collisions, uppercase and lowercase letters should never be folded together when processing environment variables.
- Environment-variable names should *never begin* with a digit (to avoid confusing some tools or applications).
- The values assigned to environment variables are not restricted except that they end in a null byte and that a system may impose a limit on the *total* number of bytes used to store values (specified in an optional environment variable, ARG_MAX, of course).

Tools

Each Unix shell provides tools (not always the same) to create an environment variable and assign (or reassign) it a value, remove that value, and report an environment variable's current value.

CREATE or SET:
In CSH, use

```
setenv VARIABLE value
```

to add the variable VARIABLE to the environment of all subsequently executed commands and to assign *value* to that variable (or to change the value of an existing environment variable). For example,

```
setenv PRINTER p280
```

In SH, use

```
export VARIABLE=value
```

to do the same (note that there are no spaces flanking the equals sign here). For example

```
export PRINTER=p280
```

In KSH, use the two commands

```
VARIABLE=value

export VARIABLE
```

together to fill the same role. For example

```
PRINTER=p280
export PRINTER
```

Note that at LLNL *on BlueGene/L only*, any environment variables set with these standard commands will *not* be visible to application programs launched using MPIRUN. MPIRUN-executed programs on BG/L can only access environment variables that you have overtly declared in the quoted, blank-delimited argument to MPIRUN's -env option. For example,

```
mpirun -env "BGLMPI_ALLREDUCE=MPICH BGL_APP_L1_WRITE_THROUGH=1" ...
```

REMOVE VALUE:
In CSH, SH, or KSH, the same command removes the previously assigned value of a declared environment variable:

```
unset VARIABLE
```

For example,

```
unset PRINTER
```

leaves PRINTER without a value. You need not UNSET a variable before assigning it a new value. In CSH only, a further command (UNSETENV) actually deletes the specified variable from the execution environment altogether:

```
unsetenv VARIABLE
```

REPORT VALUE:
In CSH, SH, or KSH, the same command displays the currently assigned value (if any) of a specified environment variable:

```
printenv VARIABLE
```

For example,

```
printenv PRINTER
```

returns the string (such as 'p280') currently stored in PRINTER. All three shells also return the environment variable's value in response to either of these uses of the ECHO command

```
echo $VARIABLE
echo "$VARIABLE"
```

while these other invocations

```
echo \$VARIABLE
echo '$VARIABLE'
```

only return the literal string $VARIABLE (e.g., $PRINTER) rather than the value currently stored in VARIABLE (e.g., p280).

Security Issues

Misuse of environment variables is a security concern on some (usually user-administered) Linux computer systems because:

- There is no length limit on the string that can be stored in an environment variable, then extracted later by some malware program for inappropriate purposes.
- Environment variables are stored as an array of pointers to character strings of the form VARIABLE=*value*. On some (poorly administered) Linux systems it is possible to insert more than one VARIABLE=*value* string (for the same variable), each with a different *value*, into that array. A program could then test on the first value but actually retrieve a very different value later.

LC systems, including those that run CHAOS (the locally enhanced version of Linux), all take general administrative precautions to avoid common security problems arising from environment variable misuse. For example, although their number seems large, LC actually limits to a modest set the universe of environment variables used by the operating system (page 10) or system-set to guide user programs (page 13), as later sections of this manual explain. So the drastic measures sometimes suggested elsewhere, such as unsetting all environment variables at the start of your session, are neither necessary nor appropriate in LC's high-performance computing environment. Likewise, the Livermore Computing Resource Management (LCRM) system, which manages batch jobs on LC production machines, carefully controls the environment variables to which each job is exposed (see the details below (page 27)). So if you use environment variables responsibly yourself, the LC software environment is unlikely to pose security problems for you based on environment variable values.

Because of its unusual role in specifying *dynamically* linked libraries, environment variable LD_LIBRARY_PATH is sometimes mentioned as as special source of security problems. For LC's policy on this variable and strategy for best use, see the "LD_LIBRARY_PATH Details" section below (page 43).

Kinds of Environment Variables at LC

A complex UNIX computing environment may involve hundreds of distinct environment variables with specific, sometimes obscure or overlapping uses. This section divides the most important environment variables on LC production machines into broad kinds based on how they are usually set or used (e.g., by the system, by you, to manage batch jobs, etc.).

System Variables

These environment variables are important for proper functioning of the operating system (AIX or CHAOS) but most user applications in most situations can ignore them.

General system variables:

ACLOCAL_PATH

AUTHSTATE

BRIDGE_CONFIG_FILE

DB2INSTANCE

DB_PROPERTY

DCE_SEC_KEYMGMT_WAKEUP_INTERVAL

DSHPATH

GNOME2_PATH

GTK_PATH

G_BROKEN_FILENAMES

IC

INFOPATH

INTEL_LICENSE_FILE

JAVA_BINDIR

JAVA_HOME

JAVA_ROOT

JRE_HOME

K5MUTE

LM_LICENSE_FILE

LSTC_LICENSE_SERVER

MAIL

MMCS_SERVER_IP

ODMDIR

PKG_CONFIG_PATH

RPC_RESTRICTED_PORTS

RPC_UNSUPPORTED_NETIFS

TKG_LMHOST

TRY_PE_SITE

VT_THREADS

VWSPATH

System variables that support Dotkit:

DK_NODE	is an ordered list of top-level directories that contain Dotkit packages.
DK_ROOT	is the directory that contains the scripts that Dotkit executes.
DK_SUBNODE	is an ordered list of subdirectories in each DK_NODE directory that contain Dotkit packages.
DK_UEQRU	(default is 1) ???
FPATH	???
_dk_inuse	(default is 0) is a list of currently invoked Dotkit packages.
_dk_isatty	???
_dk_rl	???
_dk_shell	is the user's current shell.

Get-Only Variables

These environment variables are set by system software so they can be queried by user applications (to discover the software context). But their values should *not* be changed by user programs. LC system administrators manage these environment variables for you to increase your chances of reliable production runs and to minimize potential security problems.

ENVIRONMENT	is either INTERACTIVE or BATCH, depending on whether you or LCRM is managing the session. Starting in February, 2006, LCRM will execute your batch job even if you (foolishly) change ENVIRONMENT to INTERACTIVE in your job script.
HOME	is the pathname of the user's home directory.
HOST_GRP	contains a string that all "like" LC machines share to identify their similar software context (for updates or run control during batch jobs). For example, all LC CHAOS machines, open and secure, have "linux" as their HOST_GRP. See also the relevant section (URL: http://www.llnl.gov/LCdocs/chome/index.jsp?show=s2.2) of the Common Home Reference Manual.
HOSTNAME	is the name of the user's current login node (e.g., mcr37).
LOGIN	is the sequence number of the current session (e.g., 1).
LOGNAME	is the user's login name (e.g., tomk) on this machine (same as USER but used by System-V-derived UNIX software).
NETWORK	(Tru64 machines only; not used with AIX or Linux/CHAOS) specifies the security flavor of the local network, where possible values are OCF (open network) and SCF (secure network).
SHELL	is the absolute pathname of the current shell (e.g., /bin/csh).
SSH_CLIENT	(set by OpenSSH) is a space-delimited sequence containing the IP address and port number where the SSH client runs (where the incoming SSH connection originates), followed by the port number of the SSH server that handles this connection.
SSH_CONNECTION	(set by OpenSSH) contains a space-delimited list of the IP address of the client, the client's port number, the IP address of the server, and then the server's port number (used to maintain connection security).
SSH_TTY	(set by OpenSSH) contains the absolute pathname (e.g., /dev/pts/53) of the TTY device used for the shell when your SSH connection was created.

SYS_TYPE	contains a string that all LC machines with the same (specific version of the) operating system share, to allow unambiguous testing to discover the operating context for an application. For example, MCR now has chaos_3_x86_elan3 and its SYS_TYPE. See also the relevant section (URL: http://www.llnl.gov/LCdocs/chome/index.jsp?show=s2.2) of the Common Home Reference Manual.
USER	is the user's login name (e.g., tomk) on this machine (same as LOGNAME, but used by BSD-derived UNIX software).

User-Setable Variables

These environment variables are both set and queried by user scripts and applications, usually to customize the software context. Some have "standard" default values and others are assigned a "best" local value by the system administrators on LC production machines (there are many more similar environment variables that are available to users but that LC does *not* preset to any value).

Different users with different computing needs or goals may find different value changes appropriate. For the first subgroup, user customization typically involves *appending* strings (usually pathnames) to an existing value. For the second subgroup, user customization typically involves *replacing* one value (often numeric or Boolean) with an alternative. In each group, the first few variables, set off from the main alphabetical list, are the most frequently customized by users.

Besides these general-purpose user-setable environment variables, many others are indirectly set for special task control and resource allocation purposes whenever you run SLURM's SRUN job-execution tool. Those SLRUM variables and their relation to SRUN options are explained separately below (page 37).

Append Values

Customize these user-setable environment variables by APPENDING to their existing value:

LD_LIBRARY_PATH	
	(Linux/CHAOS only; for AIX see LIBPATH) specifies a colon-delimited list of directories for the linker to check (in order) to find dynamically linked libraries. Correct use of this variable is complex; see the special comments below (page 43).
LIBPATH	(AIX only; for Linux/CHAOS see LD_LIBRARY_PATH) specifies a colon-delimited list of directories for the linker to check (in order) to find dynamically linked libraries.
PATH	specifies a colon-delimited list of directories for the shell to search, in order of preference, to find programs to execute (this is your "search path").

CLASSPATH	specifies where to find the class libaries needed for the Java Virtual Machine and other Java applications.
LOCPATH	supports C-program localization ("locale" use, which determines the allowed characters, collating sequence, and handling of time, date, monetary, and numeric strings).
MANPATH	specifies the directories for the shell to search, in order of preference, to find MAN pages to deliver.
NLSPATH	supports C-program localization ("locale" use, which determines the allowed characters, collating sequence, and handling of time, date, monetary, and numeric strings).

PERL5LIB	specifies the absolute pathname for the local Perl library (usually a child of /usr/local/lib).
XLOCALEDIR	specifies the absolute pathname of the local library that supports X11 use.

Replace Values

Customize these user-setable environment variables by REPLACING their existing value:

DISPLAY	contains the address of the local workstation where you want output from X-Windows applications to display.
TERM	specifies the type of terminal (e.g., xterm-color) for which output is needed.

AIXTHREAD_COND_DEBUG

(default is OFF) toggles keeping a list of (Pthreads) condition variables to assist with debugging threaded programs.

AIXTHREAD_MINKTHREADS

overrides the AIXTHREAD_MNRATIO environment variable (next). This allows you to manually specify the minimum number of active kernel threads (default follows from MNRATIO). The library scheduler will not reclaim kernel threads below this number.

AIXTHREAD_MNRATIO

specifies the ratio of pthreads (M) to kernel threads (N). AIXTHREAD_MNRATIO is examined when the system creates a pthread to determine if a kernel thread should also be created to maintain the correct ratio. You can set this environment variable by supplying a value of the form

```
p:k
```

where k is the number of kernel threads the system uses to handle p (user) pthreads. You may specify any positive integer for p and k, but these values are used in a formula that employs integer arithmetic and this results in the loss of some precision when big numbers are specified. (See also AIXTHREAD_MINKTHREADS, above.)

Defaults:
If k is greater than p, the ratio is treated as 1:1.
If you specify no value, the default depends on the default contention scope.
If system scope contention is the default, the ratio is 1:1.
If process scope contention is the default, the ratio in 8:1.

AIXTHREAD_MUTEX_DEBUG

(default is OFF) toggles keeping a list of (Pthreads) active mutexes to assist with debugging threaded programs.

AIXTHREAD_RWLOCK_DEBUG

(default is OFF) toggles keeping a list of (Pthreads) read-write locks to assist with debugging threaded programs.

AIXTHREAD_SCOPE

sets the contention scope of pthreads created using the default pthread attribute object (for background on contention scope, see the "Alternative Thread Implementation Models" section of the Pthreads Overview (for LC) (URL: http://www.llnl.gov/LCdocs/pthreads). You can specify either of two exclusive values for this variable:

P	indicates process scope (the default).
S	indicates system scope.

AIXTHREAD_SLRATIO

determines the number of kernel threads used to support local pthreads sleeping in the library code while awaiting a pthread event, for example, attempting to obtain a mutex (discussed in the "Synchronization" section of LC's Pthreads Overview (URL: http://www.llnl.gov/LCdocs/pthreads/index.jsp?show=s8)). The reason to maintain kernel threads for sleeping pthreads is that, when the awaited pthread event occurs, the pthread will immediately need a kernel thread to run on. Using a kernel thread that is already available is more efficient than creating a new kernel thread after the event has taken place.

You can set this environment variable by supplying a value of the form

```
    k:p
```

where k is the number of kernel threads to reserve for every p sleeping (user) pthreads. WARNING: the relative positions of k and p are reversed here from the ratio used to assign a value to AIXTHREAD_MNRATIO. You may specify any positive integer for p and k, but these values are used in a formula that employs integer arithmetic and this results in the loss of some precision when big numbers are specified. (See also AIXTHREAD_MINKTHREADS, above.)

Defaults:
If k is greater than p, the ratio is treated as 1:1.
If you specify no value, the default ratio is 1:12.

BGL_APP_L1_SWOA

(BG/L only) controls overwriting the L1 cache memory. On BG/L, each CPU has a fast, 32-kbyte local cache (L1), and both CPUs on a chip share a larger but slower prefetch cache (L2) as well as another 4-Mbyte cache (L3). BGL_APP_L1_SWOA toggles the "store-without-allocate" feature of the L1 cache as follows:

0	(default, disables SWOA) instructs each CPU to load the cache line into L1 and perform a read-modify-update. This sacrifices the former contents of L1.
1	(enables store-without-allocate, SWOA) loads the cache line into the L3 cache instead of L1. L3 performs the read-modify-update and so this preserves the contents of L1. Codes that depend heavily on the use of L1 sometimes perform much better with SWOA enabled.

BGL_APP_L1_WRITE_THROUGH

(BG/L only) controls writethrough (immediate backend copying) of the L1 cache. Jobs on BG/L that encounter a parity error in the 32-kbyte L1 cache (that is local to each CPU) promptly die. At LLNL, full-system runs have only about a 6-hour mean time to failure from such L1 errors. BGL_APP_L1_WRITE_THROUGH toggles your response to this problem as follows:

0	(default) disables L1 writethrough (copying waits for a threshold so restoring L1 from a copy usually fails). In this case, every L1 parity error kills your job.
1	enables L1 writethrough for the current job. This can be inefficient (it often incurs a 10% to 40% performance penalty), but it virtually eliminates machine stops and job death when L1 parity errors occur.

BGLMPI_COLLECTIVE_DISABLE

(BG/L only) controls which implementation is used for eight MPI collective operations (ALLGATHER, ALLGATHERV, ALLREDUCE, ALLTOALL, ALLTOALLV, BARRIER, BCAST, and REDUCE) managed *as a set*. The choices are:

0	(default) uses optimized collective routines for all eight operations (usually improves application performance).
1	disables optimization and uses MPICH routines instead for all eight operations. This can let you work around unexpected errors that seem related to collective operations, but with a performance penalty. To change implementation details for any *single* specific collective routine on BG/L, use one of the eight specific environment variables (with a shared syntax) described at BGLMPI_*COLLECTIVE* below.

BGLMPI_*COLLECTIVE*

(BG/L only) represents eight environment variables (listed below) with a common syntax whose role is to each control which implementation is used for one of eight MPI collective operations on BG/L (to toggle all eight collectives *at once*, use BGLMPI_COLLECTIVE_DISABLE above). Each BGLMPI_*COLLECTIVE* environment variable can be used in three ways:

(1) Disable Optimization.
Set the variable's value to MPICH to disable the corresponding optimized collective routine and instead invoke the "safe" MPICH algorithm for that collective operation (same effect individually that BGLMPI_COLLECTIVE_DISABLE=1 has globally).

(2) Specify a Network.
Set the variable's value to the BG/L network that you prefer for this collective operation (for example, BGLMPI_BCAST=TREE). Available network choices are GI (global interrupt), TREE, and TORUS (but not all values work for all eight environment variables). The default network for each collective operation comes first in the list assigned to it below. Some collectives accept an optional "packet inject parameter" (whose default is 3) for the TORUS value.

(3) Provide a Failover Sequence.
Leave the variable's value blank (unset) and BG/L will try each alternative *in order* in the comma-delimited list below associated with each of the eight MPI collective operations (for example, for ALLREDUCE the failover sequence will be first TREE, then TORUS, then MPICH).

The eight BG/L collective environment variables and the associated possible values for each one are:

BGLMPI_ALLGATHER={TORUS:3,MPICH}

BGLMPI_ALLGATHERV={TORUS:3,MPICH}

BGLMPI_ALLREDUCE={TREE,TORUS,MPICH}

BGLMPI_ALLTOALL={TORUS:3,MPICH}

BGLMPI_ALLTOALLV={TORUS:3,MPICH}

BGLMPI_BARRIER={GI,TREE,TORUS,MPICH}

BGLMPI_BCAST={TREE,TORUS}

BGLMPI_REDUCE={TREE,TORUS,MPICH}

BGLMPI_EAGER

(BG/L only; default is 1000) specifies (with an integer value *nnnn*) the size in bytes of the interprocess message at which BG/L switches from using the *eager* to the *rendezvous* protocol. The eager protocol sends data asynchronously and immediately to the destination (fast but sometimes problem prone), while the rendezvous protocol sends data only when requested by the destination. By default, messages sized less than or equal to 1000 bytes use the eager protocol, and messages sized 1001 bytes or greater use the rendezvous protocol.

BGLMPI_MAPPING

(BG/L only) controls the mapping of MPI processes to physical CPUs to make the most of BG/L's unusual features for point-to-point communication. The two mapping alternatives are:

XYZT	(default) maps the MPI tasks to the first CPU on each node using (x,y,z) order (for co-processor mode, where only one CPU/node computes), and then uses the same order for assigning tasks to the second CPU on each node (for virtual-node mode, where both CPUs compute).
TXYZ	maps the MPI tasks in consecutive pairs to the two CPUs on the first node, then by pairs to each other node in (x,y,z) order (so that task0 and task1 go to the two CPUs on node0, task2 and task3 go to the two CPUs on node1, etc.). This method guarantees two tasks/node in virtual-node mode, sometimes a performance enhancer.

BGLMPI_PACING

(BG/L only) toggles the use of flow control (packet pacing) for interprocess messages larger than 1000 bytes (BG/L "rendezvous messages," see also BGLMPI_EAGER). MPI applications that heavily use collectives (such as LINPACK) seldom benefit from packet pacing, but applications that send large messages point to point may see big performance gains. The choices are:

YES	(default) enables packet pacing with a "pacing window" size of 16.
NO	disables packet pacing.

BGLMPI_RVZ	(BG/L only) is the same as BGLMPI_EAGER, explained above (page 20).
BGLMPI_RZV	(BG/L only) is the same as BGLMPI_EAGER, explained above (page 20).
CHECKPOINT	(default is NO) if set to YES enables job checkpointing on AIX machines that also use SLURM instead of LoadLeveler. See the "Checkpointing with SLURM and POE" section (URL: http://www.llnl.gov/LCdocs/dpcs/index.jsp?show=s7.5) of LC's LCRM Reference Manual for full details on the correct use of this and its related variables.
CVSUMASK	specifies the file permissions (e.g., 007) assigned to new directories and their contents that are managed by the Concurrent Versions System (CVS).
CVS_RSH	specifies an external program (usually RSH or SSH) for connecting to the Concurrent Versions System server.
EDITOR	specifies the pathname of the user's preferred text editor.

LANG	(default at LC is en_US) specifies the "locale" (for localization) if all other LC_ variables are not assigned.
LC_ALL	specifies the "locale" (for localization) and overrides any other LC_ variables that may have been assigned.
LC_FASTMSG	specifies the language for localization of system messages.
LESS	allows users of the text-display tool LESS to store options for LESS (e.g., -ie), that will be invoked automatically whenever they execute LESS.
LESSOPEN	specifies a local "input preprocessor" for LESS (such as /usr/bin/lesspipe.sh), which allows LESS to display more diverse file types (including even TAR and some graphics files).
LIBELAN_GALLOC_EBASE	(default value: 0xb0000000) resizes the Elan global memory heap for MPI collective operations. EBASE is a pointer to a base virtual address in Elan memory to be used for the global heap. (Set this variable and the next two if you use MPI collectives, such as REDUCE, GATHER, SCATTER, or their ALL versions, with more than about 100 processes.)
LIBELAN_GALLOC_MBASE	(default value: 0xb0000000) resizes the Elan global memory heap for MPI collective operations. MBASE is a pointer to the main memory base in Elan memory to be used for the global heap.
LIBELAN_GALLOC_SIZE	(default value: 16777216) resizes the Elan global memory heap for MPI collective operations. SIZE is the size in bytes of the Elan global heap.
LIBELAN_WAITTYPE	(suggested value: POLL) specifies how a blocking MPI process will share computing resources (comparable to MP_WAIT_MODE under POE on IBM machines). Possible values are:

POLL	(default) has the receiving thread actively poll for incoming messages. Use this choice for all MPI jobs on clusters that have a Quadrics interconnect.
SLEEP	has the receiving thread sleep and thus remove itself from the active dispatching queue.
YIELD	has the receiving thread stay in the queue but yield the processor if it has no work to do.

LLAPIERRORMSGS

(default at LC is YES on AIX machines that use LoadLeveler instead of SLURM) toggles the display of messages from LoadLeveler about errors detected in its configuration file.

LS_COLORS (Linux/CHAOS only) specifies a colon-delmited list of codes (each a suffix-number equation) that tell LS which colors to use for which file types during display to your terminal. DIRCOLORS is the utility that manages this environment variable.

MAILMSG (default on AIX is 'You have new mail') is the message displayed when new mail arrives on this system.

MALLOC_TRIM_THRESHOLD

(suggested value: -1) see the next item for joint use.

MALLOC_MMAP_MAX

(suggested value: 0) when combined, MALLOC_TRIM_THRESHOLD and MALLOC_MMAP_MAX force MALLOC to use SBRK() rather than MMAP() to allocate memory. This improves performance, but it may reduce the total amount of memory available to your user processes (to no more than 1 Gbyte/process).

MANPAGER specifies the pager (text display program) used when you invoke MAN. MANPAGER, if set, overrides the value of PAGER, which MAN uses otherwise.

MP_CKPTDIR (only on AIX machines that also use SLURM instead of LoadLeveler) specifies the absolute pathname of the directory to receive checkpoint files. See also MP_CKPTFILE.

MP_CKPTFILE (only on AIX machines that also use SLURM instead of LoadLeveler) overrides MP_CKPTDIR and specifies an absolute pathname for checkpoint files. See the "Checkpointing with SLURM and POE" section (URL: http://www.llnl.gov/LCdocs/dpcs/index.jsp?show=s7.5) of LC's LCRM Reference Manual for the full story on using these variables correctly for job checkpointing.

MP_COREFILE_FORMAT

(coresponds to the POE -corefile_format flag on AIX machines) specifies the format for corefiles generated when any of your processes terminate abnormally (this variable is especially suited for managing parallel programs). Possible values are:

[unset]	generates a standard AIX corefile for abnormal termination.
STDERR	sends the corefile contents to standard error for abnormal termination.
any.other.string	generates a lightweight corefile for abnormal termination. The string CORE.LIGHT, and hence this format alternative, is the default on LC's AIX machines.

MP_COREFILE_SIGTERM

(default is NO) prevents unexpected and undesirable core dumps that otherwise occur when you kill a parallel job (using PRM) or when a parallel job exits by calling MPI_Abort. Value YES allows such core dumps.

MP_CPU_USE

(interactive jobs only) specifies whether or not your job is willing to share a node's CPU with other jobs, where the choices are UNIQUE (no node sharing, the default at LC and in general for nonspecific node allocation of US communication jobs) or MULTPLE (the nonLC default for IP jobs).

MP_EUILIB specifies which of two protocols should be used for task communications, where the choices are:

US	selects User Space protocol, the default on LC SP systems, and the faster of the two choices.
IP	selects Internet Protocol.

MP_HOSTFILE

(not used on LC SP systems) specifies the name of a file that contains the domain names of each node to use during specific node allocation (set to NULL by default at LC).

MP_INFOLEVEL

specifies the level of messages reported as an aid to debugging, where the choices are
0 (error messages only),
1 (warning and error, the default at LC),
2 (informational, warning, and error),
3 (above plus diagnostic),
4, 5, 6 (above plus more elaborate diagnostic messages used by the IBM Support Center).

MP_LABELIO

specifies whether or not output from the parallel tasks is labeled by taskid, where the choices are YES (the default at LC) or NO.

MP_RESD	(interactive jobs only) specifies whether or not LoadLeveler (or SLURM on PU) or the individual user should allocate nodes to this job (ignored for batch jobs), where the choices are:

YES	has LoadLeveler allocate the nodes (called nonspecific node allocation, this is the default at LC).
NO	has the user allocate the nodes (called specific node allocation, not used on LC machines).

MP_RMLIB	specifies the IBM resource management library (so for LC AIX machines that use SLURM instead of LoadLeveler to manage resources, MP_RMLIB is overtly unset).
MP_SHARED_MEMORY	specifies whether tasks running on the same node should use shared memory (yes, the default) or the SP switch (no) for MPI message passing. Shared memory is faster for some codes, but the overhead may decrease the performance of others.
OBJECT_MODE	distinguishes (values are either 32 or 64) among AIX systems with the same SYS_TYPE but different (32-bit or 64-bit) executables (for compiler selection).
PAGER	(default is MORE) specifies the prefered text-file display tool.
PS1	specifies the string used as your primary shell prompt.
PSTAT_CONFIG	(no default) stores a comma-delimited list of arguments (field names) for PSTAT's -o option, so you can easily, repeatedly ask for the same customized job status report when you use PSTAT to check on a batch job. For an alphabetical, explanatory list of allowed field names for -o (and hence for PSTAT_CONFIG), see the "Run Properties of Batch Jobs" section (URL: http://www.llnl.gov/LCdocs/dpcs/index.jsp?show=4.2b) of LC's LCRM Reference Manual.
SPINLOOPTIME	(no default) specifies the number of times that the system will try to get a busy lock without taking a secondary action, such as calling the kernel to yield the processor. Manipulating SPINLOOPTIME can be helpful on SMP systems, where the lock might be held by another actively running pthread and will soon be released. On uniprocessor systems this value is ignored.
TERMCAP	supports character-cell terminals and printers (retained only for backward compatibility with some older programs).

TMP	specifies the pathname of a temporary directory (often the same as TMPDIR) for application programs.
TMPDIR	(default is /usr/tmp) specifies the pathname of a temporary directory for utility intermediate files (such as the temporary files made when SORT or HTAR runs).
TZ	specifies the local time zone (US/Pacific).
XCURSOR_THEME	controls the contrast and transparency of the cursor (and, in some applications, also the text) displayed on X-windows terminals.
YIELDLOOPTIME	(no default) specifies the number of times that the system yields the processor when trying to acquire a busy mutex or spin lock (see the Synchronization (URL: http://www.llnl.gov/LCdocs/pthreads/index.jsp?show=s8) section of the Pthreads Overview manual for details) before going to sleep on the lock. YIELDLOOPTIME can be helpful for complex applications where multiple locks are in use.

Batch-Job Environment Variables

The Livermore Computing Resource Management (LCRM) system uses environment variables (including some discussed in previous sections, but not limited to those) to manage your batch jobs and to run those jobs successfully on your target machine. LCRM divides the batch-relevant environment variables into four disjoint sets (and you can optionally set others and pass their values to your job when you submit it if you wish).

The first subsection below (page 27) identifies, lists the members of, and briefly explains the four default sets of LCRM-relevant environment variables. The second subsection below (page 34) then refers to those sets when it tells how LCRM handles various environment variables at job submittal and later at job execution (including the order in which their values are assigned for each batch job run). Environment variables used by SLURM, and set by SLURM's SRUN utility, serve to control tasks and their resources "below LCRM," within each LC cluster environment. Hence, they are (mostly) listed and explained in a separate subsection (page 37) of their own.

Environment Variables Known to LCRM

For every batch job that LCRM manages, it interacts with four disjoint sets of environment variables as follows (independent of any others declared by each job's owner):

```
A. SUBMITTAL environment variables--
   Those that LCRM creates from your job and its
   environment when you run PSUB.
B. EXECUTION environment variables--
   Those that LCRM sets for your job in the job's
   execution environment.
C. UNSET environment variables--
   Those that LCRM unsets for your job
   when the job executes.
D. DEPRECATED environment variables--
   Formerly important for LCRM but now replaced
   by others in sets A, B, or C.
```

Each subsection below explains one of these four sets, names each environment variable in that set (alphabetically), and briefly describes its role for LCRM.

A. Submittal Variables

LCRM creates these environment variables from your job and its submittal environment on the machine where you execute PSUB. The goal is to preserve values of environment variables on the submitting host (that is, key features of the job's submittal environment) because these may change as the job runs, or may be different in the job's *executing* environment. Some additional submittal variables contain LCRM-specific data about the job (such as its ID string). All of these variables (and hence their values) *are* passed to your running job by LCRM.

PSUB_DEP_JOBID	is the LCRM identification number for the job (if any) on which this job depends.
PSUB_HOME	is set to $HOME on the host where you ran PSUB.
PSUB_HOST	contains the name of the host where you ran PSUB.
PSUB_JOBID	is the LCRM job identifier.
PSUB_LOGNAME	is set to $LOGNAME on the host where you ran PSUB.
PSUB_PATH	is set to $PATH on the host where you ran PSUB.
PSUB_REQNAME	contains the (specified or implied) job name.
PSUB_SUBDIR	contains the name of the directory where you ran PSUB on the submittal host.
PSUB_TZ_ENV	is set to $TZ on the host where you ran PSUB.
PSUB_USER	is set to $USER on the host where you ran PSUB.
SESSARGS	contains the arguments that you used when you ran PSUB to submit your job.

B. Execution Variables

LCRM automatically sets these environment variables for your job when the job starts in your specified *execution* (not submittal) environment. (To set others as well, see the discussion of PSUB's -x option in the next section (page 34).

Some of these LCRM execution variables are normally set by the operating system for any shell (HOME, LOGIN, USER). See the "Get-Only Variables" section (page 13) above. Others are normally set by each user (PATH, TZ). See the "User-Setable Variables" section (page 15) above. Still others have special roles (SLURM-JOBID, MPIRUN_PARTITION) and are useful if and only if specific job-support software (such as SLURM) comes into play.

ENVIRONMENT	is always set to the uppercase string BATCH. Starting in February, 2006, LCRM will run your batch job even if your script (foolishly) resets ENVIRONMENT to INTERACTIVE.
HOME	specifies your home directory on the execution host.
LOADL_ACTIVE	is the current version of LoadLeveler on the execution host (if LoadLeveler is used instead of SLURM).
LOADLBATCH	is set to YES to help LoadLeveler avoid conflicts with other environment variables that it and your job might share.
LOGIN	is your LC user name (same as LOGNAME, and different from the interactive use of this variable to hold the sequence number (e.g., 1) of this login session).
LOGNAME	is your LC user name (same as USER but used by System-V-derived UNIX software).
LCRM_SERIESID	stores this job's "series ID," used internally by LCRM to distinguish different jobs that end up with the same job ID when the ID numbers have "wrapped around" and started to repeat.
LRM_GLOBAL_SESSION_ID	is the machine and process ID of this job's "leader" process.
MPIRUN_NOALLOCATE	helps manage MPI jobs on BlueGene/L, where it is set to TRUE.
MPIRUN_NOFREE	is set to YES (to TRUE on BlueGene/L), same as using -nofree on the MPIRUN execute line.

MPIRUN_PARTITION

supplies block information to MPIRUN, which is used to launch parallel tasks successfully on BlueGene/L.

PATH is set to /bin:/usr/bin by default (which you can alter or expand with other pathnames).

PCS_TMPDIR specifies the location of a temporary directory that LCRM creates when your job starts, that persists during the whole job, and that is automatically purged when the job completes. System administrators configure the directory name, or disable it.

PSUB_WORKDIR

contains the same value as PCS_TMPDIR if that environment variable is set. Otherwise, contains the pathname of your home directory ($HOME) on the *execution* machine.

RMS_RESOURCEID

contains the Resource Management System (RMS) ID for this batch job (useful only on machines that still use RMS, such as LANL's Q machine).

SHELL is the pathname of the shell that interprets the job script.

SLURM_JOBID

is the job ID of this batch script's underlying SLURM job (if any). This can be reset by SLURM's SRUN utility.

SLURM_NETWORK

(On AIX machines only, LCRM automatically sets this extra environment variable when SLURM is used instead of LoadLeveler) specifies four network features for each SLURM job step (which under AIX means for each POE invocation), using an argument with this sequential format:
network.[*protocol*],[*device*],[*adapteruse*],[*mode*]
where:

protocol specifies the network protocol (such as MPI).

device specifies the kind of switch used for communication (ethernet, FDDI, etc.), where the choices are the same abbreviation strings as the possible values of environment variable MP_EUIDEVICE (see the POE User Guide, "Task Communication" section (URL: http://www.llnl.gov/LCdocs/poe/index.jsp?show=s6.3.3)).

adapteruse specifies whether (SHARED) or not (DEDICATED) your job is willing to share a node's switch adapter with other jobs (see the POE User Guide, "Other POE Environment Variables" section (URL: http://www.llnl.gov/LCdocs/poe/index.jsp?show=s6.4) on corresponding environment variable MP_ADAPTER_USE).

mode	specifies which of two protocols or modes should be used for task communications, where the choices are the same as the possible values of environment variable MP_EUILIB (see "User Setable Variables" above (page 15)).

SLURM_NNODES

is the actual number of nodes assigned by SLURM to run your job (this may exceed the node count that you requested with SRUN's -N option). See the "SRUN Resource-Allocation Options" section (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.4) in the SLURM Reference Manual for details.

SLURM_NPROCS

is the total number of processes run for your job by SLURM. This is analogous to MP_PROCS as used by POE on AIX machines, except that SLURM_NPROCS applies to batch jobs as well. See also the "Comparison with POE" section (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.2) of the SLURM Reference Manual for more details.

SLURM_UID	identifies the user to SLURM.
TZ	specifies your job's local time zone (US/Pacific for LC machines). You can reset this with SRUN.
USER	is your LC user name (same as LOGNAME but used by BSD-derived UNIX software).

C. Unset Variables

LCRM unsets all of these environment variables in the *execution* environment, when your job starts to run (even if you use PSUB's -x option):

KRB5CCNAME	supports Kerberos with a directory name for KINIT use.
LD_LIBRARY_PATH	(Linux/CHAOS only) specifies an ordered, colon-deliminted list of directories for the linker to check to find dynamically loaded libraries. See the detailed explanation below (page 43).
RMS_MEMLIMIT	supports Resource Management System (RMS), no longer used on LC machines.
RMS_PARTTION	supports Resource Management System (RMS), no longer used on LC machines.
RMS_PRIORITY	supports Resource Management System (RMS), no longer used on LC machines.
RMS_TIMELIMIT	supports Resource Management System (RMS), no longer used on LC machines.
TERM	specifies a terminal type (for interactive use only).
TERMCAP	supports character-cell terminals and printers (retained only for backward compatibility with some older programs).

D. Deprecated Variables

These environment variables once filled an important role for LCRM, but now each has been superceded by a replacement variable with a different name (and hence already explained by that name in subsections A, B, or C above).

LCRM_REQID	deprecated. This will be replaced by PSUB_JOBID in a version of LCRM after version 6.14.
PCS_REQID	deprecated. This will be replaced by PSUB_JOBID in a version of LCRM after version 6.14.
PSUB_SHELL	deprecated. This will be eliminated completely in a version of LCRM after version 6.14 (it contains the user's login shell on the *submittal* rather than the execution machine).
QSUB_HOST	deprecated. This will be replaced by PSUB_HOST in a version of LCRM after version 6.14.
QSUB_REQID	deprecated. This will be replaced by PSUB_JOBID in a version of LCRM after version 6.14.
QSUB_REQNAME	deprecated. This will be replaced by PSUB_REQNAME in a version of LCRM after version 6.14.
QSUB_WORKDIR	deprecated. This will be replaced by PSUB_WORKDIR in a version of LCRM after version 6.14.

How LCRM Uses Environment Variables

LCRM interacts with relevant environment variables (I) when you submit your batch job and (II) again later when your job starts in its execution environment. This section explains *which* environment variables LCRM uses at each phase (job submittal, execution) and also the *order* in which those variables are set (or reset) by all of the software that can affect them in either phase. This explanation refers to the four groups of LCRM-relevant environment variables defined and enumerated in the previous subsection (page 27).

At Job Submittal

THE SEQUENCE:
Here is the sequence in which environment variables (that may affect your batch job) are set on the machine where you *submit* the job (that is, where you run PSUB):

Submittal
Sequence (I)

(1) The system sets the usual array of "system" (page 10) and "get-only" (page 13) environment variables when you log in.

(2) The shell SOURCEs your dot files, which may in turn set additional environment variables (page 15) (such as PATH).

(3) You may set (unset, reset) other environment variables interactively.

(4) You run PSUB.

- (a) Some (but usually not all) aspects of (1), (2), and (3) are captured in the specific members of LCRM's "Group A" (PSUB) environment variables (page 28).
- (b) You decide whether to also pass the remaining values to your job by invoking/omitting PSUB's -x option (described below).

WITHOUT -x:
If you run PSUB and omit the -x (lowercase eks) option, then LCRM passes all the "Group A" environment variables (page 28) from the submittal environment, unsets all the "Group C" environment variables (page 32), and sets all the "Group B" environment variables (page 29) on the machine where your job runs when your job starts. However, *no* other environment variables that you may have defined in steps (2) or (3) above before your ran PSUB are passed to your job. The default (omitting -x) leaves your job's execution environment somewhat independent of its submittal environment.

WITH -x:
If you run PSUB and invoke the -x option, then LCRM passes all the "Group A" environment variables (page 28) from the submittal environment, unsets all the "Group C" environment variables (page 32), and sets all the "Group B" environment variables (page 29) on the machine where your job runs when your job starts. LCRM *also* passes to the job the values of any additional environment variables that you set (or

reset) on the submittal machine in your dot files (step 2) or interactively (step 3) prior to running PSUB. Thus, using -x lets you heavily influence your job's execution environment by sharing with it customizing environment-variable decisions that you made earlier, even if they were not represented in "Group A" or "Group B" above. There are, however, two exceptions (see the next paragraph).

EXCEPTIONS:
Even if you invoke PSUB's -x option when you submit your batch job, LCRM handles two environment variables in a special way.

ENVIRONMENT

is always set to the uppercase string BATCH on your job's execution machine regardless of its value on the submittal machine (your job script can then reliably test for this value as it runs). As a safety measure, starting in February, 2006, LCRM version 6.14 will execute your batch job even if your script (foolishly) resets ENVIRONMENT to INTERACTIVE.

LD_LIBRARY_PATH

is always *unset* on your job's execution machine regardless of its value on the submittal machine. You should set it within your job script if it needs a special value. Correct use of this variable on LC Linux/CHAOS machines is complicated by several interdependencies. See the detailed explanation below (page 43).

At Job's Execution

Here is the sequence in which environment variables (that may affect your batch job) are set on the machine where your job *runs* when it starts (which may be quite different than the machine on which you executed PSUB to submit the job):

Execution Sequence (II)

(1) LCRM spawns the shell for your batch job.

(2) LCRM (un)sets the environment variables relevant to your specific job.

- (a) LCRM passes the "Group A" general environment variables (page 28) to the execution environment.
- (b) LCRM sets the "Group B" general environment variables (page 29).
- (c) IF and ONLY IF you invoked the -x option when you submitted this job with PSUB, LCRM then sets or resets any additional "customized" environment variables that you specified in your dot files or interactively on the *submittal* machine before you ran PSUB (but that were outside Groups A and B). By default, LCRM omits this step.

(3) The system sets the usual array of "system" (page 10) and "get-only" (page 13) environment variables when you log in.

(4) The shell SOURCEs your dot files, which may in turn set additional environment variables (page 15) (such as PATH).

(5) Your job script executes, optionally setting additional environment variables by means of commands within it.

The last step here (5) is your job's chance to

- *read* any of the Group A or B variables as already set above, or
- *test* previously set environment variables (such as ENVIRONMENT, which LCRM always sets to BATCH) to trigger special conditional script actions, or
- *set* previously unset environment variables (such as LD_LIBRARY_PATH, which LCRM always unsets) to meet your job's special local needs in its execution environment.

SLURM Environment Variables

The Simple Linux Utility for Resource Management (SLURM) handles resource allocation and task management on the level below LCRM, on individual high-performance clusters. This section introduces the environment variables that SLURM sets and uses as it manages jobs. For general information on SLURM and its specific tools, such as SRUN, see LC's SLURM Reference Manual (URL: http://www.llnl.gov/LCdocs/slurm).

SRUN Option Variables

Many SLURM SRUN options have corresponding environment variables (analogous to the approach used with POE). Each SRUN option, if invoked during execution, always overrides (resets) the corresponding environment variable (which contains each job feature's default value, if there is a default). For more background on SRUN and how to use it, or for an explanation of any SRUN option listed in the table below, consult the SRUN section (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2) of the SLURM Reference Manual.

Environment Variable	Corresponding SRUN Option(s)
SLURM_ACCOUNT	-U, --account
SLURM_CPU_BIND	--cpu_bind
SLURM_CPUS_PER_TASK	-c, --ncpus-per-task
SLURM_CONN_TYPE	--conn-type
SLURM_CORE_FORMAT	--core-format
SLURM_DEBUG	-v, --verbose
SLURMD_DEBUG	-d, --slurmd-debug
SLURM_DISABLE_STATUS	-X, --disable-status
SLURM_DISTRIBUTION	-m, --distribution
SLURM_GEOMETRY	-g, --geometry
SLURM_LABELIO	-l, --label
SLURM_MEM_BIND	--mem_bind
SLURM_NETWORK(*)	--network
SLURM_NNODES	-N, --nodes
SLURM_NO_ROTATE	--no-rotate
SLURM_NPROCS	-n, --ntasks
SLURM_OVERCOMMIT	-o, --overcommit
SLURM_PARTITION	-p, --partition
SLURM_REMOTE_CWD	-D, --chdir
SLURM_SRUN_COMM_IFHN	--ctrl-comm-ifhn
SLURM_STDERRMODE	-e, --error
SLURM_STDINMODE	-i, --input
SLURM_STDOUTMODE	-o, --output
SLURM_TASK_EPILOG	--task-epilog
SLURM_TASK_PROLOG	--task-prolog
SLURM_TIMELIMIT	-t, --time
SLURM_WAIT	-W, --wait

(*)See explanatory details below (page 42).

Task-Environment Variables

In addition to the option variables cited above (page 37), SRUN sets these environment variables (a few are the same as option variables) for each executing task on each remote compute node (under any operating system).

SLURM_CPU_BIND_VERBOSE

affects the reporting of CPU/task binding, as explained in the "Affinity or NUMA Constraints" section (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.7.2) of the SLURM Reference Manual under --cpu_bind.

SLURM_CPU_BIND_TYPE

affects the binding of CPUs to tasks, as explained in the "Affinity or NUMA Constraints" section (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.7.2) of the SLURM Reference Manual under --cpu_bind.

SLURM_CPU_BIND_LIST

affects the binding of CPUs to tasks, as explained in the "Affinity or NUMA Constraints" section (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.7.2) of the SLURM Reference Manual under --cpu_bind.

SLURM_CPUS_ON_NODE

specifies the number of processors available to the job on this node.

SLURM_JOBID

specifies the job ID of the executing job (see also SRUN's --jobid option (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.5.2)).

SLURM_LAUNCH_NODE_IPADDR

specifies the IP address of the node from which the task launch initiated (the node where SRUN executed).

SLURM_LOCALID

specifies the node-local task ID for the process within a job.

SLURM_MEM_BIND_VERBOSE

affects the reporting of memory/task binding, as explained in the "Affinity or NUMA Constraints" section (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.7.2) of the SLURM Reference Manual under --mem_bind.

SLURM_MEM_BIND_TYPE

affects the binding of memory to tasks, as explained in the "Affinity or NUMA Constraints" section (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.7.2) of the SLURM Reference Manual under --mem_bind.

SLURM_MEM_BIND_LIST

affects the binding of memory to tasks, as explained in the "Affinity or NUMA Constraints" section (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.7.2) of the SLURM Reference Manual under --mem_bind.

SLURM_NNODES

is the actual number of nodes assigned to run your job (which may exceed the number of nodes that you explicitly requested with SRUN's -N option (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.4)).

SLURM_NODEID

specifies the relative node ID of the current node.

SLURM_NODELIST

specifies the list of nodes on which the job is actually running.

SLURM_NPROCS

specifies the total number of processes in the job.

SLURM_PROCID

specifies the MPI rank (or relative process ID) for the current process.

SLURM_TASKS_PER_NODE

specifies the number of tasks to initiate on each node. Values are a comma-delimited list in the same order as SLRUM_NODELIST. To specify two or more nodes with the same task count, follow the count by (x#), where # is the repetition count. For example,
SLURM_TASKS_PER_NODE=2(x3),1
indicates two tasks per node on the first three nodes, then one task on the fourth node.

MPIRUN_PARTITION

(BlueGene/L only) specifies the block name.

MPIRUN_NOALLOCATE

(BlueGene/L only) prevents allocating a block.

MPIRUN_NOFREE

(BlueGene/L only) prevents freeing a block.

Other SLURM-Relevant Variables

Other environment variables important for SLURM SRUN-managed jobs include:

MAX_TASKS_PER_NODE

provides an upper bound on the number of tasks that SRUN assigns to each job node, even if you allow more than one process per CPU by invoking SRUN's -O (uppercase oh) option (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.5.3).

SLURM_HOSTFILE

names the file that specifies how to assign tasks to nodes, rather than using the block or cyclic approaches toggled by SRUN's -m (--distribution) option (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s4.2.5.1).

On AIX (IBM) machines only, this extra environment variable is automatically set by LCRM when SLURM is used instead of LoadLeveler (alternatively, set with SRUN's --network=*type* option even though POE launches tasks instead of SRUN under AIX):

SLURM_NETWORK

specifies four network features for each SLURM job step (which under AIX means for each POE invocation), using an argument with this sequential format:
network.[*protocol*],[*device*],[*adapteruse*],[*mode*]
where:

protocol	specifies the network protocol (such as MPI).
device	specifies the kind of switch used for communication (ethernet, FDDI, etc.), where the choices are the same abbreviation strings as the possible values of environment variable MP_EUIDEVICE (see the POE User Guide, "Task Communication" section (URL: http://www.llnl.gov/LCdocs/poe/index.jsp?show=s6.3.3)).
adapteruse	specifies whether (SHARED) or not (DEDICATED) your job is willing to share a node's switch adapter with other jobs (see the POE User Guide, "Other POE Environment Variables" section (URL: http://www.llnl.gov/LCdocs/poe/index.jsp?show=s6.4) on corresponding environment variable MP_ADAPTER_USE).
mode	specifies which of two protocols or modes should be used for task communications, where the choices are the same as the possible values of environment variable MP_EUILIB (see the POE User Guide, "Task Communication" section (URL: http://www.llnl.gov/LCdocs/poe/index.jsp?show=s6.3.3)).

LD_LIBRARY_PATH Details

Environment variable LD_LIBRARY_PATH (on Linux/CHAOS machines only) specifies a colon-delimited, ordered list of directories (pathnames) to search first for *dynamically* linked libraries, before the standard set of Linux library directories. This is especially useful for testing a new, nonstandard, or debug-version of a library. (Dynamically linked libraries are those not incorporated into your binary file at link time, but instead referenced by their location. All have .so suffixes, such as libnew.so.) On AIX systems, environment variable LIBPATH fills the same role as LD_LIBRARY_PATH does under Linux.

Using LD_LIBRARY_PATH correctly on LC machines calls for careful attention to several interdependencies. Invoking the setup script for the Intel compilers (ICC, IFC) on the SCF Linux clusters, namely

```
    source /opt/intel/compiler50/ia32/bin/iccvars.sh
```

sets the environment variable LD_LIBRARY_PATH to specific Intel libraries. If you alternate between using ICC and the GNU compiler GCC, you should unset this variable by hand each time you use GCC. Note that sometimes Linux system updates leave LD_LIBRARY_PATH undefined, so your scripts should always test for this possibility before trying to append (or prepend) other path values to whatever this environment variable contains.

Under CHAOS 3.0 (deployed late in 2005), the compiler scripts link with -rpath for the MPI libraries by default. Consequently:

(1) If you previously set LD_LIBRARY_PATH to /usr/lib/mpi/lib or /usr/lib/mpi/mpi_gnu/lib by hand, this step is no longer needed.

(2) If you need "side installs" to handle MPI-vendor bugs, you should link with -rpath=no and then set LD_LIBRARY_PATH to your preferred nondefault directory.

(3) Remember that like all environment variables, LD_LIBRARY_PATH is inherited by all programs called by the one for which you set it. This may sometimes produce unintended errors or problems for child processes.

(4) Remember also that LCRM *unsets* LD_LIBRARY_PATH on the execution machine at the start of every batch job that it manages. If, for any of the above reasons, your batch job needs a specific value in LD_LIBRARY_PATH, your own job script must provide that value.

Dictionary of LC Environment Variables

Find the environment variable that interests you in this alphabetical list, then follow the online link or printed-page cross reference to read the passage that describes it in this manual.

Dotkit

Details coming soon.

Disclaimer

Keyword Index

To see an alphabetical list of keywords for this document, consult the next section (page 51).

Alphabetical List of Keywords

Keyword	Description
a	The alphabetical index of keywords.
append	Customize by appending value.
availability	Where these programs run.
background	Environment var. basic characteristics.
batch-variables	Used (mostly) to manage batch jobs.
date	The latest changes to this document.
dictionary	Alphabetical summary of LC env. vars.
dotkit	Managing env. vars. with Dotkit.
entire	This entire document.
get-only-variables	Set by system, used by users.
index	The structural index of keywords.
introduction	Overview of this document.
kinds	Kinds of LC environment variables.
lcrm-deprecated-variables	Formerly important LCRM env. vars.
lcrm-ev-sets	Environment variables known to LCRM.
lcrm-execution-usage	LCRM use at batch-job execution.
lcrm-execution-variables	Execution-machine environment set up.
lcrm-submittal-usage	LCRM use at batch-job submittal.
lcrm-submittal-variables	Submittal-machine environment capture.
lcrm-unset-variables	Environment vars unset by LCRM.
lcrm-variable-usage	How LCRM uses environment variables.
ld-library-path	LD_LIBRARY_PATH use at LC explained.
other-slurm	Other SLURM-relevant variables.
replace	Customize by replacing value.
revisions	The complete revision history.
roles	Environment versus other variables.
scope	Topics covered in this document.
security	Known env-var security issues.
slurm-variables	How SLURM uses environment variables.
srun-options	Vars corresponding to SLURM SRUN options.
srun-task-variables	Vars that SRUN sets for each task.
syntax	IEEE syntax standards for env. vars.
system-variables	Set and used by system only.
title	The name of this document.
tools	How to set and show env. vars.
user-setable-variables	Set and used by users (mostly).
who	Who to contact for assistance.

Date and Revisions

Revision Date	Keyword Affected	Description of Change
14Sep06	slurm-variables	Added section, 3 subsections on SLURM.
	user-setable-variables	Added cross ref to SLURM sections.
	dictionary	SLURM variables added to list.
	index	New keywords for new sections.
07Jun06	user-setable-variables	Section subdivided for better access.
		Sixteen BG/L specialized variables added.
	index	New keywords for new sections.
	tools	MPIRUN -env role on BG/L noted.
	dictionary	User get/set variables alphabetized.
13Mar06	lcrm-submittal-usage	New BATCH/INTERACTIVE value independence.
	lcrm-execution-variables	LCRM_SERIESID new with LCRM 6.14.
		SLURM_NPROCS compared with MP_PROCS.
	user-setable-variables	PSTAT_CONFIG added, explained.
		CHECKPOINT, MP_CKPTDIR, MP_CKPTFILE added and explained.
08Feb06	ld-library-path	New explanatory section added.
	index	New keyword for new section.
	tools	Many details added.
	security	Many details added.
	get-only-variables	NETWORK role explained.
	user-setable-variables	Seven variables added.
		Eleven explanations expanded.
	lcrm-execution-variables	More details on SLURM variables.
18Jan06	entire	First edition of Environment Vars. manual.

TRG (14Sep06)